# Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play

Sainbayar Sukhbaatar [1] Ilya Kostrikov [1] Arthur Szlam [2] Rob Fergus [1 2]

## Abstract

We describe a simple scheme that allows an agent to explore its environment in an unsupervised manner. Our scheme pits two versions of the same agent, Alice and Bob, against one another. Alice proposes a task for Bob to complete; and then Bob attempts to complete the task. In this work we will focus on (nearly) reversible environments, or environments that can be reset, and Alice will "propose" the task by running a set of actions and then Bob must partially undo, or repeat them, respectively. Via an appropriate reward structure, Alice and Bob automatically generate a curriculum of exploration, enabling unsupervised training of the agent. When deployed on an RL task within the environment, this unsupervised training reduces the number of episodes needed to learn.

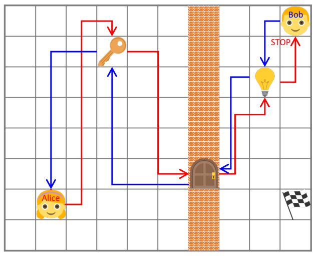

*Figure 1.* Illustration of self-play in a Mazebase task. Alice sets Bob a task he must complete. In this instance, Alice first picks up the key, then opens the door. After passing through, she turns off the light and then decides to perform the STOP action. This hands control of the agent over to Bob. He must now return the environment to its original state to receive the (internal) reward. Thus he must first turn the light back on, then go back through the door and drop the key at its original location before returning to Alice's start location. This forces Bob to learn the function of the various items in the environment. This task is just one of many devised by Alice, who automatically builds a curriculum of increasingly challenging tasks. When Bob is presented with a new evaluation task, e.g. go to the flag (present only in the new task) to obtain external reward, he is able to learn it relatively quickly since he already knows how the environment works.

## 1. Introduction

Model-free approaches to reinforcement learning are inefficient, typically requiring a huge number of episodes to learn a satisfactory policy. The lack of an explicit environment model means the agent must learn it from scratch at the same time as it tries to understand which trajectories lead to rewards. In environments where reward is sparse, only a small fraction of the agents' experience is directly used to update the policy, contributing to the inefficiency.

In this paper we introduce a novel form of unsupervised training for an agent that enables exploration and learning about the environment without any external reward that incentivizes the agents to learn how to transition between states as efficiently as possible. We demonstrate that this unsupervised training allows the agent to learn new tasks within the environment more quickly than starting with a random policy.

[1]Dept. of Computer Science, Courant Institute, New York University [2]Facebook AI Research, New York. Correspondence to: Sainbayar Sukhbaatar<sainbar@cs.nyu.edu>.

## 2. Approach

We consider environments with a single physical agent, but we allow it to have two separate minds: Alice and Bob, each with its own objective and parameters. At a high level, Alice's job is to propose a task for Bob to complete, and Bob's job is to complete the task.

Our approach is restricted to two classes of environment: (i) those that are (nearly) reversible, or (ii) ones that can be reset to their initial state at least once. These restrictions allow us to sidestep complications around how to communicate the task and determine its difficulty (see Section 5 for further discussion). In these two scenarios, Alice starts at some initial state $s_0$ and proposes a task by *doing* it, i.e. executing a sequence of actions that takes the agent to a state $s_t$. She then outputs a STOP action which hands control over to Bob. In reversible environments, Bob's goal is to return the agent back to state $s_0$ (or within some margin of it, if the state is continuous), to receive reward. In partially observable environments, the objective is relaxed to Bob finding a state that returns the same observation as Alice's initial state. In environments where resets are permissible, Alice's STOP action also reinitializes the environment, thus Bob starts at $s_0$ and now must reach $s_t$ to be rewarded, thus repeating Alice's task instead of reversing it. See Fig. 1 for an example, and also Algorithm 1.

In both cases, this self-play between Alice and Bob only involves internal reward (detailed below), thus multiple rounds can be performed without needing any supervisory signal from the environment. As such, it comprises a form of unsupervised training where Alice and Bob explore the environment and learn how it operates. This exploration can be leveraged for some target task by using Bob's policy as the agent's initialization. Alternatively, the self-play and target task episodes can be interleaved, biasing the exploration to be in service of the target task.

We choose the reward structure for Alice and Bob to encourage Alice to push Bob past his comfort zone, but not give him impossible tasks. Denoting Bob's reward by $R_b$ and Alice's reward by $R_a$, we use

$$R_b = -t_b \tag{1}$$

where $t_b$ is the time taken by Bob to complete his task (and is set it set maximum value $t_b = t_{\text{Max}}$ if Bob fails) and

$$R_a = \max(0, t_b - t_a) \tag{2}$$

where $t_a$ is the time until Alice performs the STOP action. Thus Alice is rewarded if Bob takes more time, but the negative term on her own time will encourage Alice not to take too many steps when Bob is failing. For both reversible and resettable environments, Alice must limit her steps to make Bob's task easier, thus Alice's optimal behavior is to find simplest tasks that Bob cannot complete. This eases learning for Bob since the new task will be only just beyond his current capabilities. The self-regulating feedback between Alice and Bob allows them to automatically construct a curriculum for exploration, a key contribution of our approach.

### 2.1. Parameterizing Alice and Bob's actions

Alice and Bob each have policy functions which take as input two observations of state variables, and output a distribution over actions . In Alice's case, the function will be of the form

$$a_{\text{Alice}} = f_A(s_t, s_0),$$

where $s_0$ is the observation of the initial state of the environment and $s_t$ is the observation of the current state. In Bob's case, the function will be

$$a_{\text{Bob}} = f_B(s'_t, s'_0),$$

where $s'_0 = s_0$ when we have a reversible environment. In a resettable environment $s'_0$ is the state where Alice executed the stop action. Note that the "observations" can include a parameterized model of a raw observation. When a target task is presented, the agent's policy function is $a_{\text{Target}} = f_B(s''_t, e)$, where $e$ is a special observation corresponding to the target task.

In the experiments below, we demonstrate our approach in settings where $f$ is tabular; where it is a neural network taking discrete inputs, and where it is a neural network taking in continuous inputs.

### 2.2. Universal Bob in the tabular setting

We now show that in environments with finite states, tabular policies, and Markovian transitions, we can interpret the reset and reverse games as training the agents to find policies that can get from any state to any other in the least expected number of steps.

Note that as discussed above, the policy table for both Alice and Bob is indexed by $(s_i, s_0)$ or $(s_i, s_T)$, not just by $s_i$. In particular, with the assumptions above, this means that there is a policy $\pi_{\text{fast}}$ such that $\pi_{\text{fast}}(s_0, s_T)$ has the smallest expected number of steps to transition from $s_0$ to $s_T$. Call any such policy a *fast policy*. It is clear that $\pi_{\text{fast}}$ is a universal policy for Bob, such that for any Alice policy $\pi_a$, $\pi_{\text{fast}}$ is optimal with respect to $\pi_a$. In a reset game, with deterministic transitions, $\pi_{\text{fast}}$ nets Alice a return of 0, and in the reverse game, the return of $\pi_{\text{fast}}$ against an optimal Bob also using $\pi_{\text{fast}}$ can be considered a measure of the reversibility of the environment.

For this discussion, assume that we are using the reset game or the reverse game in a perfectly reversible environment. If $\pi_A$ and $\pi_B$ are policies of Alice and Bob that are in equilibrium (that is, one cannot make Alice better without changing Bob, and one cannot make Bob better without changing Alice), $\pi_B$ is a fast policy. To see this, note that if $\pi_B$ is not fast, then we can replace it with $\pi_{\text{fast}}$, and then for any challenge $(s_0, s_T)$ that Alice gives Bob with nonzero probability and for which $\pi_{\text{fast}}(s_0, s_T)$ gives a smaller number of expected steps, Bob will get a higher reward. On the other hand, if Alice is not giving positive probability to some challenge $(s_0, s_T)$ (where the initial probability of Alice starting at $s_0$ is nonzero), and if Bob's policy on $(s_0, s_T)$ is not fast, then Alice can use $\pi_{\text{fast}}(s_0, s_T)$ and increase her reward.

Thus we can see that in the finite, tabular, and Markovian setting, the asymmetric self-play can be interpreted as a method for training Alice and Bob to be able to transit between pairs of states as efficiently as possible.

## 3. Related Work

Self-play arises naturally in reinforcement learning, and has been well studied. For example, for playing checkers (Samuel, 1959), backgammon (Tesauro, 1995), and Go, (Silver et al., 2016), and in in multi-agent games such as RoboSoccer (Riedmiller et al., 2009). Here, the agents or teams of agents compete for external reward. This differs from our scheme where the reward is purely internal and the self-play is a way of motivating an agent to learn about its environment to augment sparse rewards from separate target tasks.

```
function SelfPlayEpisode(Reverse/Repeat, $t_{\text{Max}}$)
    $t_a \leftarrow 0$
    $s_0 \leftarrow$ env.observe()
    $s_{\text{Target}} = s_0$
    while True do
        # Alice's turn
        $t_a \leftarrow t_a + 1$
        $s_{t_a} \leftarrow$ env.observe()
        $a_{t_a} \leftarrow$ policy.Alice($s_{t_a}$, $s_0$)
        if $a_{t_a}$ = STOP then
            $s_{\text{Target}} = s_{t_a}$
            env.reset()
            break
        env.act($a_{t_a}$)
    $t_b \leftarrow 0$
    while True do
        # Bob's turn
        $t_b \leftarrow t_b + 1$
        $s'_{t_b} \leftarrow$ env.observe()
        $a'_{t_b} \leftarrow$ policy.Bob($s'_{t_b}$, $s_{\text{Target}}$)
        env.act($a'_{t_b}$)
        if $s'_{t_b} = s_{\text{Target}}$ or $t_b > t_{\text{Max}}$ then
            break
    $R_a = \max(0, t_b - t_a)$
    $R_b = -t_b$
    policy.Alice.update($R_a$)
    policy.Bob.update($R_b$)
    return
```

Our approach has some relationships with Generative adversarial networks (GANs) (Goodfellow et al., 2014), which train a generative neural net by having it try to fool a discriminator network which tries to differentiate samples from the training examples. (Li et al., 2017) introduce an adversarial approach to dialogue generation, where a generator model is subjected to a form of "Turing test" by a discriminator network. (Mescheder et al., 2017) demonstrate how adversarial loss terms can be combined with variational auto-encoders to permit more accurate density modeling. While GAN's are often thought of as methods for training a generator, the generator can be thought of as a method for generating hard negatives for the discriminator. From this viewpoint, in our approach, Alice acts as a "generator", finding "negatives" for Bob, and Bob is the "discriminator".

There is a large body of work on intrinsic motivation (Barto, 2013; Singh et al., 2004; Klyubin et al., 2005; Schmidhuber, 1991) for self-supervised learning agents. These works propose methods for training an agent to explore and become proficient at manipulating its environment without necessarily having a specific target task, and without a source of extrinsic supervision. One line in this direction is curiosity-driven exploration (Schmidhuber, 1991). These techniques can be applied in encouraging exploration in the context of reinforcement learning, for example (Bellemare et al., 2016; Strehl & Littman, 2008; Lopes et al., 2012; Tang et al., 2016); Roughly, these use some notion of the novelty of a state to give a reward. In the simplest setting, novelty can be just the number of times a state has been visited; in more complex scenarios, the agent can build a model of the world, and the novelty is the difficulty in placing the current state into the model. In our work, there is no explicit notion of novelty. Even if Bob has seen a state many times, if he has trouble getting to it, Alice should force him towards that state. Another line of work on intrinsic motivation is a formalization of the notion of empowerment (Klyubin et al., 2005), or how much control the agent has over its environment. Our work is related in the sense that it is in both Alice's and Bob's interests to have more control over the environment; but we do not explicitly measure that control except in relation to the tasks that Alice sets.

Curriculum learning (Bengio et al., 2009) is widely used in many machine learning approaches. Typically however, the curriculum requires at least some manual specification. A key point about our work is that Alice and Bob devise their own curriculum entirely automatically. Previous automatic approaches, such as (Kumar et al., 2010), rely on monitoring training error. But since ours is unsupervised, no training labels are required either.

Our basic paradigm of "Alice proposing a task, and Bob doing it" is related to the Horde architecture (Sutton et al., 2011) and (Schaul et al., 2015). In those works, instead of using a value function $V = V(s)$ that depends on the current state, a value function that explicitly depends on state and goal $V = V(s,g)$ is used. In our experiments, our models will be parameterized in a similar fashion. The novelty in this work is in how Alice defines the goal for Bob.

## 4. Experiments

The following experiments explore our self-play approach on a variety of tasks, both continuous and discrete, from the Mazebase (Sukhbaatar et al., 2015) and RLLab (Duan et al., 2016) environments. The same protocol is used in all settings: self-play and target task episodes are mixed together and used to train the agent via discrete policy gradient. We evaluate both the reverse and repeat versions of self-play. We demonstrate that the self-play episodes help training, in terms of number of target task episodes needed to learn the task (where we consider the self-play episodes "free", since they make no use of environmental reward).

In all the experiments we use use policy gradient (Williams, 1992) with a baseline for optimizing the policies. In the tabular task below, we use a constant baseline; in all the other tasks we use a policy parameterized by a neural network, and a baseline that depends on the state. Denote the states in an episode by $s(1),...,s(T)$, and the actions taken at each of those states as $a(1),...,a(T)$, where $T$ is the length of the episode. The baseline is a scalar function of the states $b(s,\theta)$, computed via an extra head on the network producing the action probabilities. Beside maximizing the expected reward with policy gradient, the models are also trained to minimize the distance between the baseline value and actual reward. Thus after finishing an episode, we update the model parameters $\theta$ by

$$\Delta\theta = \sum_{t=1}^{T}\left[\frac{\partial \log p(a(t)|s(t),\theta)}{\partial\theta}\left(\sum_{i=t}^{T} r(i) - b(s(t),\theta)\right) - \lambda\frac{\partial}{\partial\theta}\left(\sum_{i=t}^{T} r(i) - b(s(t),\theta)\right)^2\right]. \quad (3)$$

Here $r(t)$ is reward given at time $t$, and the hyperparameter $\gamma$ is for balancing the reward and the baseline objectives, which set to 0.1 in all experiments.

## 4.1. Long hallway

We first describe a simple toy designed to illustrate the function of the asymmetric self-play. The environment consists of $M$ states $\{s_1,...,s_M\}$ arranged in a chain. Both Alice and Bob have three possible actions, "left", "right", or "stop". If the agent is at $s_i$ with $i \neq 1$, "left" takes it to $s_{i-1}$; "right" analogously increases the state index, and "stop" transfers control to Bob when Alice runs it and terminates the episode when Bob runs it. We use "return to initial state" as the self-play task (i.e. Reverse in Algorithm 1). For the target task, we randomly pick a starting state and target state, and the episode is considered successful if Bob moves to the target state and executes the stop action before a fixed number of maximum steps.

In this case, the target task is essentially the same as the self-play task, and so running it is not unsupervised learning (and in particular, on this toy example unlike the other examples below, we do not mix self-play training with target task training). However, we see that the curriculum afforded by the self-play is efficient at training the agent to do the target task at the beginning of the training, and is effective at forcing exploration of the state space as Bob gets more competent.

In Fig. 2 we plot the number of episodes vs rate of success at the target task with four different methods. We set $M = 25$ and the maximum allowed steps for Alice and Bob to be 30. We use fully tabular controllers; the table is of size $M^2 \times 3$, with a distribution over the three actions for each possible (start, end pair).

The red curve corresponds to policy gradient, with a penalty of $-1$ given upon failure to complete the task, and a penalty of $-s/30$ for successfully completing the task in $s$ steps. The magenta curve corresponds to taking Alice to have a random policy (1/2 probability of moving left or right, and not stopping till the maximum allowed steps). The green curve corresponds to policy gradient with an exploration bonus similar to (Strehl & Littman, 2008). That is, we keep count of the number of times $N_s$ the agent has been in each state $s$, and the reward for $s$ is adjusted by exploration bonus $\alpha/\sqrt{N_s}$, where $\alpha$ is a constant balancing the reward from completing the task with the exploration bonus. We choose the weight $\alpha$ to maximize success at 0.2M episodes from the set $\{0,0.1,0.2,...,1\}$. The blue curve corresponds to the asymmetric self-play training.

We can see that at the very beginning, a random policy for Alice gives some form of curriculum but eventually is harmful, because Bob never gets to see any long treks. On the other hand, policy gradient sees very few successes in the beginning, and so trains slowly. Using the self-play method, Alice gives Bob easy problems at first (she starts from random), and then builds harder and harder problems as the training progresses, finally matching the performance boost of the count based exploration. Although not shown, similar patterns are observed for a wide range of learning rates.

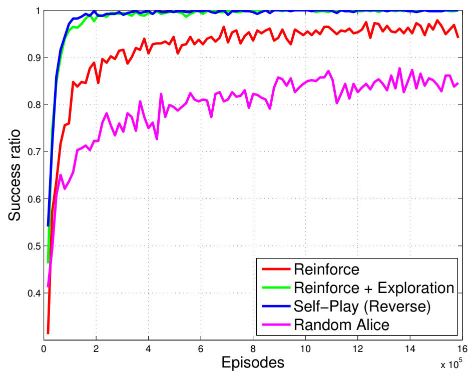

*Figure 2.* The hallway task from section 4.1. The $y$ axis is fraction of successes on the target task, and the $x$ axis is the total number of training examples seen. Standard policy gradient (red) learns slowly. Adding an explicit exploration bonus (Strehl & Littman, 2008) (green) helps significantly. Our self-play approach (blue) gives similar performance however. Using a random policy for Alice (magenta) drastically impairs performance, showing the importance of self-play between Alice and Bob.

## 4.2. Mazebase

We now describe experiments using the MazeBase environment (Sukhbaatar et al., 2015). These have discrete actions and states, but sufficient combinatorial complexity that tabular methods cannot be used. They consist of various items placed on a finite 2D grid; the environment is randomly generated for each episode.

For both self-play and the target task, we use an environment where the maze contains a light switch, a key and a wall with a door (see Fig. 1). An agent can open or close the door by toggling the key switch, and turn on or off light with the light switch. When the light is off, the agent can only see the (glowing) light switch. There is also a goal flag item in the target task.

In self-play, an episode starts with Alice in control, who can navigate through the maze and change the switch states until she outputs the STOP action. Then, Bob takes control and tries to return everything to its original state, restricted to visible items (e.g. if light was off initially, then Bob does not need to worry about the state of door because it was invisible) in the reverse self-play. In the repeat version, the maze resets back to its initial state when Bob takes the control, who tries to reach the final state of Alice.

In the target task, the agent and the goal are always placed on opposite sides of the wall. Also, the light and key switches are placed on the same side as the agent, but the light is always off and the door is closed initially. Therefore, in order to succeed, the agent has to turn on the light, toggle the key switch to open the door, pass through it, and reach the goal flag. Reward of -0.1 is given at every step until the agent reaches the goal or episode runs more than $t_{\text{Max}} = 80$ time steps.

In self-play, episodes are also limited to $t_{\text{Max}} = 80$ time steps, and reward is only given at the end of the episode. Alice and Bob's reward from Equ. (1) and (2) is scaled by hyperparameter $\gamma = 0.1$ to match the target task reward.

Both Alice and Bob's policies are modeled by a fully-connected neural network with two hidden layers each with 100 and 50 units (with tanh non-linearities) respectively. The encoder into each of the networks takes a bag of words over (objects, locations); that is, there is a separate word in the lookup table for each (object, location) pair. As described above, $f$ takes as input two states; these are combined after the shared encoder layer by concatenation. Action probabilities are output by a linear layer followed by a softmax. In addition, the model also outputs a baseline value using a linear layer, which is trained with mean-square loss to predict the cumulative reward. The parameters of Alice and Bob are not shared.

Training used RMSProp (Tieleman & Hinton, 2012) with learning rate of 0.003 and batch size 256. All parameters are randomly initialized from $\mathcal{N}(0,0.2)$. We also use an entropy regularization term on the softmax output, set to 0.003. During each training episode, we randomly pick between self-play and target tasks with 80% and 20% probabilities respectively unless otherwise specified. Fig. 3 shows details of a single training run, demonstrating how Alice and Bob automatically build a curriculum between themselves though self-play.

### 4.2.1. Biasing for or against self-play

The effectiveness of our approach depends in part on the similarity between the self-play and target tasks. One way to explore this in our environment is to vary the probability of the light being off initially during self-play episodes[1]. Note that the light is always off in the target task; if the light is usually on at the start of Alice's turn in reverse, for example, she will learn to turn it off, and then Bob will be biased to turn it back on. On the other hand, if the light is usually off at the start of Alice's turn in reverse, Bob is strongly biased against turning the light on, and so the test task becomes especially hard. Thus changing this probability gives us some way to adjust the similarity between the two tasks.

[1]The initial state of the light should dramatically change the behavior of the agent: if it is on then agent can directly proceed to the key.

In Fig. 4, we set p(Light off)=0.5 during self-play and evaluate both reverse and repeat forms of self-play, alongside two baselines: (i) target task only training (i.e. no self-play) and (ii) self-play with a random policy for Alice. We see that the repeat form of self-play succeeds quickly while target task-only training takes much longer[2]. The reverse form of self-play and random Alice work comparably well, being in between the other two in terms of speed.

Fig. 5 shows what happens when p(Light off)=0.3. Here reverse self-play works well, but repeat self-play does poorly. As discussed above, this flipping, relative to Fig. 4, can be explained as follows: low p(Light off) means that Bob's task in reverse self-play will typically involve returning the light to the on position (irrespective of how Alice left it), the same function that must be performed in the target task. The opposite situation applies for repeat self-play, where Bob needs to encounter the light typically in the off position to help him with the test task.

In Fig. 6 we systematically vary p(Light off) between 0.1 and 0.9. The y-axis shows the speed-up (reduction in target task

[2]Training was stopped for all methods except target-only at $5 \times 10^6$ episodes.

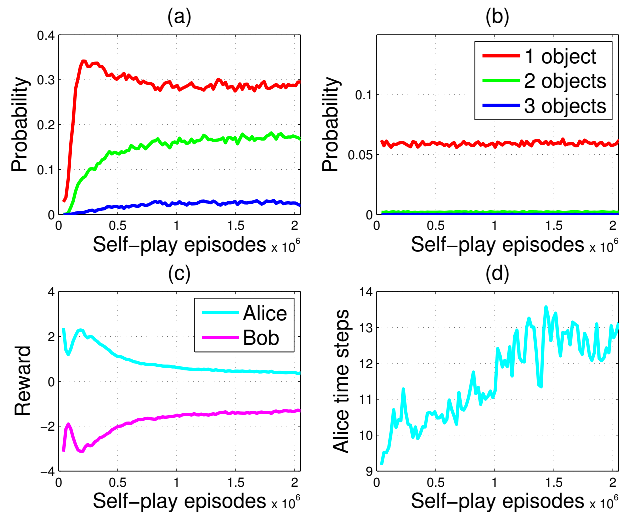

*Figure 3.* Inspection of a Mazebase learning run, using the environment shown in Fig. 1. (a): rate at which Alice interacts with 1, 2 or 3 objects during an episode, illustrating the automatically generated curriculum. Initially Alice touches no objects, but then starts to interact with one. But this rate drops as Alice devises tasks that involve two and subsequently three objects. (b) by contrast, in the random Alice baseline, she never utilizes more than a single object and even then at a much lower rate. (c) plot of Alice and Bob's reward, which strongly correlates with (a). (d) plot of $t_a$ as self-play progresses. Alice takes an increasing amount of time before handing over to Bob, consistent with tasks of increasing difficulty being set.

episodes) relative to training purely on the target-task for runs where the reward goes above -2. Unsuccessful runs are given a unity speed-up factor. The curves show that when the self-play task not biased against the target task it can help significantly.

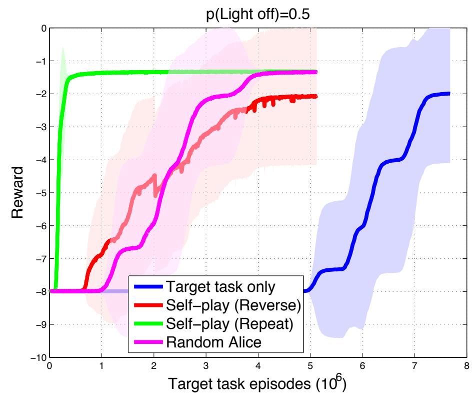

*Figure 4.* Mazebase task, illustrated in Fig. 1, for p(Light off) = 0.5. Augmenting with the repeat form of self-play enables significantly faster learning than training on the target task alone and random Alice baselines. See text and Fig. 6 for an explanation of why reverse self-play does relatively poorly. Here x-axis shows number of supervised target task episodes. As they are unsupervised, self-play episodes do not count towards this total

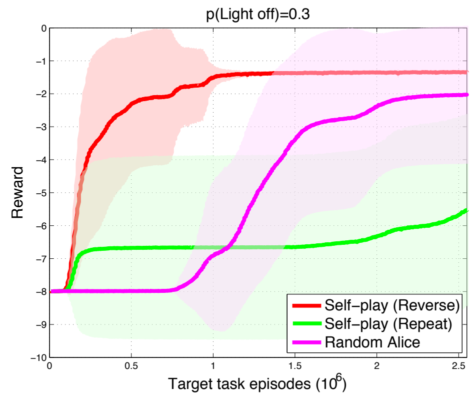

*Figure 5.* Similar to Fig. 4, but for p(Light off) set to 0.3. Here the reverse form of self-play works well. See text for details.

## 4.3. RLLab: Mountain Car

We now apply our approach to the Mountain Car task in RLLab. Here the agent controls a car trapped in a 1-D valley. It must learn to build momentum by alternately moving to the left and right, climbing higher up the valley walls until it is able to escape. Although the problem is presented as continuous, we discretize the 1-D action space into 5 bins (uniformly sized) enabling us to use discrete policy gradient, as above. An observation of state $s_t$ consists of the location and speed of the car.

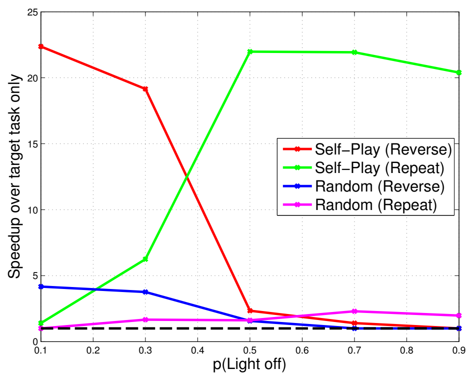

*Figure 6.* Reduction in target task episodes relative to training purely on the target-task as the distance between self-play and the target task varies (for runs where the reward goes above -2 on the Mazebase task – unsuccessful runs are given a unity speed-up factor). The $y$ axis is the speedup, and $x$ axis is p(Light off). For reverse self-play, the low p(Light off) corresponds to having self-play and target tasks be similar to one another, while the opposite applies to repeat self-play. For both forms, significant speedups are achieved when self-play is similar to the target tasks, but the effect diminishes when self-play is biased against the target task.

As in (Houthooft et al., 2016; Tang et al., 2016), a reward of +1 is given only when the car succeeds in climbing the hill, and episodes are limited to $t_{\text{Max}} = 500$ steps. We use the same hyperparameters as the Mazebase experiments, except the batch size is 128 and self-play rewards are scaled by $\gamma = 0.01$ since episodes are much longer. During training, only 1% of episodes are from the target task. In self-play, an episode is considered completed only when $\|s_b - s_a\| < 0.2$, where $s_a$ and $s_b$ are the final states of Alice and Bob respectively.

The nature of the environment makes it highly asymmetric from Alice and Bob's point of view, since it is far easier to coast down the hill to the starting point that it is to climb up it. Hence we exclusively use the reset form of self-play. In Fig. 7, we compare this to current state-of-the-art methods, namely VIME (Houthooft et al., 2016) and SimHash (Tang et al., 2016). Our approach (blue) performs comparably to both of these. We also tried using policy gradient directly on the target task samples, but it was unable to solve the problem. In this setting the self-play and target tasks are very similar thus our approach's success is somewhat expected.

## 4.4. RLLab:Swimmer

Finally, we applied our approach to the SwimmerGather task in RLLab (which uses the Mujoco (Todorov et al., 2012) simulator), where the agent controls a worm with two flexible joints, swimming in a 2D viscous fluid. In the target task, the agent gets reward +1 for eating green apples and -1 for touching red bombs, which are not present during self-play. Thus the self-play task and target tasks are different: in the former, the

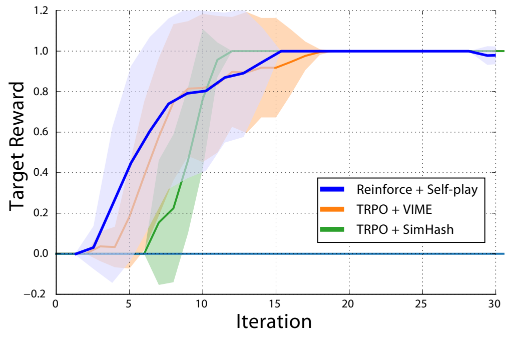

*Figure 7.* A comparison of our self-play approach on MountainCar task with VIME (Houthooft et al., 2016) and SimHash (Tang et al., 2016) (figure adapted from (Tang et al., 2016)). We plot mean rewards against the number of target task training steps (1 iter=5k steps), excluding self-play training steps as they are unsupervised. Error bars of $\pm 1\sigma$ are shown, using 10 runs of our approach. The task is fairly straightforward, being quickly mastered by all three approaches, which have similar performance. We also tried training on directly this task with Reinforce (i.e. no self-play) but found that it was unable to get any reward, despite the long training time.

worm just swims around but in the latter it must learn to swim towards green apples and away from the red bombs.

The observation state consists of a 13-dimensional vector describing location and joint angles of the worm, and a 20 dimensional vector for sensing nearby objects. The worm takes two real values as an action, each controlling one joint. We add secondary action head to our models to handle this. As in the mountain car, we discretize the output space (each joint is given 9 uniformly sized bins) to allow the use of discrete policy gradients.

The episode length is 500 steps for target tasks as in (Houthooft et al., 2016; Tang et al., 2016), and 600 for self-play. In our experiments we skip two frames with each action, but still count them toward the episode length. The hyperparameters are the same as MountainCar, except the entropy regularization is only applied to the self-play episodes and batch size is 256. Also, the self-play terminates when $\|l_b - l_a\| < 0.3$ where $l_a$ and $l_b$ are the final locations of Alice and Bob respectively. Target tasks constitute 10% of the training episodes. Fig. 8 shows the target task reward as a function of training iteration for our approach alongside VIME (Houthooft et al., 2016) and SimHash (Tang et al., 2016). Ours can be seen to gain reward earlier than the others, although it converges to a similar final value to SimHash. A video of our worm performing the test task can be found at `https://goo.gl/Vsd8Js`.

In Fig. 9 shows details of a single training run. The changes in Alice's behavior, observed in Fig. 9(c) and (d), correlate with Alice and Bob's reward (Fig. 9(b)) and, initially at least, to the reward on the test target (Fig. 9(a)). In Fig. 10 we visualize for a single training run the locations where Alice hands over to Bob at different stages of training, showing how the distribution varies.

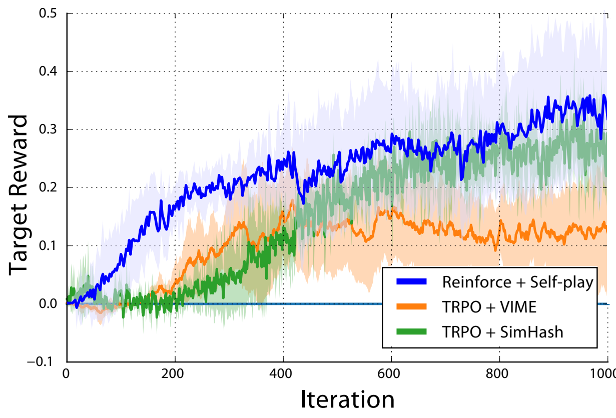

*Figure 8.* Evaluation on SwimmerGather target task, comparing to VIME (Houthooft et al., 2016) and SimHash (Tang et al., 2016) (figure adapted from (Tang et al., 2016)). Error bars are $\pm 1\sigma$ over 10 runs. With reversible self-play we are able to learn faster than the other approaches, although it converges to a comparable reward. Note that X-axis did not include self-play training steps as they are unsupervised. Training directly on the target task using Reinforce without self-play resulted in total failure.

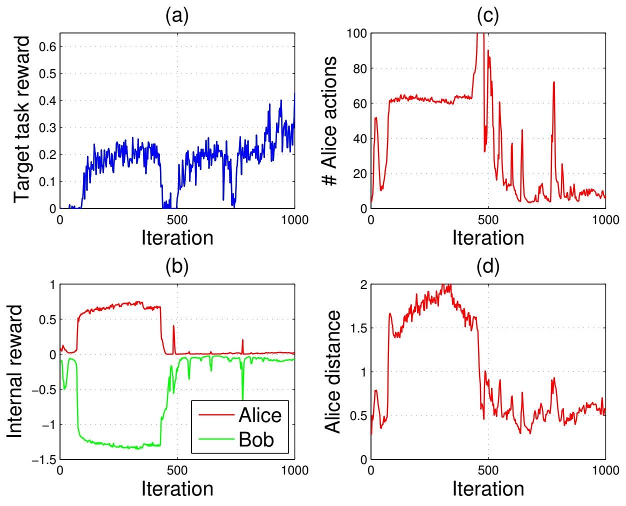

*Figure 9.* A single SwimmerGather training run. (a): Rewards on target task. (b): Rewards from reversible self-play. (c): The number of actions taken by Alice. (d): Distance that Alice travels before switching to Bob.

## 5. Discussion

In this work we described a novel method for intrinsically motivated learning which we call asymmetric self-play. Despite the method's conceptual simplicity, we have seen that it can be effective in both discrete and continuous input settings with function approximation, for encouraging exploration and automatically generating curriculums. When evaluated on challenging benchmarks, our approach is comparable to current

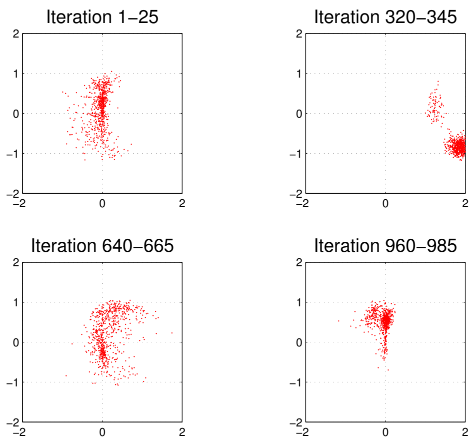

*Figure 10.* Plot of Alice's location at time of STOP action for the SwimmerGather training run shown in Fig. 9, for different stages of training. Note how Alice's distribution changes as Bob learns to solve her tasks.

state-of-the-art RL methods that incorporate an incentive for exploration. Furthermore, it is possible show theoretically that in simple environments, using asymmetric self-play with reward functions from (1) and (2), optimal agents can transit between any pair of reachable states as efficiently as possible.

However, there are limitations in the simple scheme we have described; these suggest avenues for further work:

### 5.1. Meta-exploration for Alice

We want Alice and Bob to explore the state (or state-action) space, and we would like Bob to be exposed to many different tasks. Because of the form of the standard reinforcement learning objective (expectation over rewards), Alice only wants to find the single hardest thing for Bob, and is not interested in the space of things that are hard for Bob. In the fully tabular setting, with fully reversible dynamics or with resetting, and without the constraints of realistic optimization strategies, we saw in section 2.2 that this ends up forcing Bob and Alice to learn to make any state transition as efficiently as possible. However, with more realistic optimization methods or environments, and with function approximation, Bob and Alice can get stuck in sub-optimal minima.

For example, let us follow the argument in the third paragraph of 2.2, and assume that Bob and Alice are at an equilibrium (and that we are in the tabular, finite, Markovian setting), but now we can only update Bob's and Alice's policy locally. By this we mean that in our search for a better policy for Bob or Alice, we can only make small perturbations, as in policy gradient algorithms. In this case, we can only guarantee that Bob runs a fast policy on challenges that Alice has non-zero probability of giving; but there is no guarantee that Alice will cover all possible challenges. With function approximation instead of tabular policies, we can not make any guarantees at all.

Another example with a similar outcome but different mechanism can occur using the reverse game in an environment without fully reversible dynamics. In that case, it could be that the shortest expected number of steps to complete a challenge $(s_0, s_T)$ is longer than the reverse, and indeed, so much longer that Alice should concentrate all her energy on this challenge to maximize her rewards. Thus there could be equilibria with Bob matching the fast policy only for a subset of challenges even if we allow non-local optimization.

The result is that Alice can end up in a policy that is not ideal for our purposes. In figure 10 we show the distributions of where Alice cedes control to Bob in the swimmer task. We can see that Alice has a preferred direction. Ideally, in this environment, Alice would be teaching Bob how to get from any state to any other efficiently; but instead, she is mostly teaching him how to move in one direction.

One possible approach to correcting this is to have multiple Alices, regularized so that they do not implement the same policy. More generally, we can investigate objectives for Alice that encourage her to cover a wider distribution of behaviors.

### 5.2. Communicating via actions

In this work we have limited Alice to propose tasks for Bob by doing them. This limitation is practical and effective in restricted environments that allow resetting or are (nearly) reversible. It allows a solution to three of the key difficulties of implementing the basic idea of "Alice proposes tasks, Bob does them": parameterizing the sampling of tasks, representing and communicating the tasks, and ensuring the appropriate level of difficulty of the tasks. Each of these is interesting in more general contexts. In this work, the tasks have incentivized efficient transitions. One can imagine other reward functions and task representations that incentivize discovering statistics of the states and state-transitions, for example models of their causality or temporal ordering, cluster structure.

## References

Barto, Andrew G. *Intrinsic Motivation and Reinforcement Learning*, pp. 17–47. Springer Berlin Heidelberg, 2013.

Bellemare, Marc G., Srinivasan, Sriram, Ostrovski, Georg, Schaul, Tom, Saxton, David, and Munos, Rémi. Unifying count-based exploration and intrinsic motivation. In *NIPS*, pp. 1471–1479, 2016.

Bengio, Yoshua, Louradour, Jérôme, Collobert, Ronan, and Weston, Jason. Curriculum learning. In *ICML*, pp. 41–48, 2009.

Duan, Yan, Chen, Xi, Houthooft, Rein, Schulman, John, and Abbeel, Pieter. Benchmarking deep reinforcement learning for continuous control. In *ICML*, 2016.

Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. Generative adversarial nets. In *NIPS*, pp. 2672–2680, 2014.

Houthooft, Rein, Chen, Xi, Duan, Yan, Schulman, John, Turck, Filip De, and Abbeel, Pieter. Curiosity-driven exploration in deep reinforcement learning via bayesian neural networks. *arXiv 1605.09674*, 2016.

Klyubin, Alexander S., Polani, Daniel, and Nehaniv, Chrystopher L. Empowerment: a universal agent-centric measure of control. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC*, pp. 128–135, 2005.

Kumar, M. P., Packer, Benjamin, and Koller, Daphne. Self-paced learning for latent variable models. In *NIPS*. 2010.

Li, Jiwei, Monroe, Will, Shi, Tianlin, Ritter, Alan, and Jurafsky, Dan. Adversarial learning for neural dialogue generation. *arXiv 1701.06547*, 2017.

Lopes, Manuel, Lang, Tobias, Toussaint, Marc, and Oudeyer, Pierre-Yves. Exploration in model-based reinforcement learning by empirically estimating learning progress. In *NIPS*, pp. 206–214, 2012.

Mescheder, Lars M., Nowozin, Sebastian, and Geiger, Andreas. Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks. *arXiv abs/1701.04722*, 2017.

Riedmiller, Martin, Gabel, Thomas, Hafner, Roland, and Lange, Sascha. Reinforcement learning for robot soccer. *Autonomous Robots*, 27(1):55–73, 2009.

Samuel, Arthur L. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

Schaul, Tom, Horgan, Dan, Gregor, Karol, and Silver, David. Universal value function approximators. In *ICML*, pp. 1312–1320, 2015.

Schmidhuber, J. Curious model-building control systems. In *Proc. Int. J. Conf. Neural Networks*, pp. 1458–1463. IEEE Press, 1991.

Silver, David, Huang, Aja, Maddison, Chris J., Guez, Arthur, Sifre, Laurent, van den Driessche, George, Schrittwieser, Julian, Antonoglou, Ioannis, Panneershelvam, Veda, Lanctot, Marc, Dieleman, Sander, Grewe, Dominik, Nham, John, Kalchbrenner, Nal, Sutskever, Ilya, Lillicrap, Timothy, Leach, Madeleine, Kavukcuoglu, Koray, Graepel, Thore, and Hassabis, Demis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587): 484–489, January 2016.

Singh, Satinder P., Barto, Andrew G., and Chentanez, Nuttapong. Intrinsically motivated reinforcement learning. In *NIPS*, pp. 1281–1288, 2004.

Strehl, Alexander L. and Littman, Michael L. An analysis of model-based interval estimation for markov decision processes. *J. Comput. Syst. Sci.*, 74(8):1309–1331, 2008.

Sukhbaatar, Sainbayar, Szlam, Arthur, Synnaeve, Gabriel, Chintala, Soumith, and Fergus, Rob. Mazebase: A sandbox for learning from games. *arXiv 1511.07401*, 2015.

Sutton, Richard S., Modayil, Joseph, Delp, Michael, Degris, Thomas, Pilarski, Patrick M., White, Adam, and Precup, Doina. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *AAMAS '11*, pp. 761–768, 2011.

Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. #exploration: A study of count-based exploration for deep reinforcement learning. *arXiv abs/1611.04717*, 2016.

Tesauro, Gerald. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–68, 1995.

Tieleman, T. and Hinton, G. Lecture 6.5 - rmsprop, coursera: Neural networks for machine learning, 2012.

Todorov, Emanuel, Erez, Tom, and Tassa, Yuval. Mujoco: A physics engine for model-based control. In *IROS*, pp. 5026–5033. IEEE, 2012.

Williams, Ronald J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*, pp. 229–256, 1992.